The Report Committee for Vincent Anthony Davis
Certifies that this is the approved version of the following report:

Internet Security for Mobile Computing

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Christine Julien

Sarfraz Khurshid

Internet Security for Mobile Computing

by

Vincent Anthony Davis, B.S.E.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2017

Dedication

This report is dedicated to my wife, mother, brother, and nephew. To my gorgeous wife, Adelina, your love, patience, and optimism gives me hope for better future and family. The world is yours. To my beautiful mother, Hope, no one deserves the life that you have endured. Your survival and sacrifice gives me motivation to wake up every morning. To my amazing brother, Michael, you are the best brother in the world. Lastly to my nephew, Sebastian, dream big!

Acknowledgements

I would like to express my thanks to Dr. Christine Julien for supervising this report. I would like to thank Dr. Sarfraz Khurshid for being the reader of this report.

Abstract

Internet Security for Mobile Computing

Vincent Anthony Davis, M.S.E.

The University of Texas at Austin, 2017

Supervisor: Christine Julien

Mobile devices are now the most dominant computer platform. Every time a mobile web application accesses the internet, the end user's data is susceptible to malicious attacks. For instance, when paying a bill at a store with NFC mobile payment, navigating through a city operating GPS on a smartphone, or dictating the temperature at a household with a home automation device. These activities seem routine, yet, when vulnerabilities are present they can leave holes for hackers to access bank accounts, pinpoint a user's recent location, or tell when someone is not at home. The awareness of the end user cannot be trusted. Device vendors and developers must provide safeguards.

An ongoing issue is that the present security standards are outdated and were never envisioned with mobile devices in mind. It can be suggested that security is only idling the progress of mobile computing. Still, many application developers and IT professionals do not adopt security standards fast enough to keep up-to-date with known vulnerabilities.

The main goals of the next generation of security standards, TLS, will provide developers with greater security efficiency and improved mobile throughput. These proposed capabilities of the TLS protocol will streamline mobile computing into the forefront of security practices. The analysis of this report demonstrates concepts on the direction mobile security, usability, and performance from a development standpoint.

Table of Contents

List of Tables

List of Figures

1.0 INTRODUCTION

1.1 INTRODUCTION

With the explosion of the mobile devices, security has never been more at the forefront of the technology sector. Mobile devices provide better personal services than the traditional desktop computer, even under limitations and constraints of the platform. This includes store payments, GPS navigation, storing airline boarding passes, etc. However, mobile devices present numerous security challenges, because people trust the devices with much more personal data that comes with the ease of convenient services.

Now that mobile internet browsing has surpassed that of desktop computers [1], the security industry should shift the focus on mobile devices. Transport Layer Security (TLS) is the main cryptographic and security mechanism for mobile applications that connect with the internet [4]. Unfortunately, the current iteration of TLS, version 1.2, was not envisioned for mobile environments. TLS 1.3 is the new standard that is currently in the draft state and is being built with the main objective of security and mobile performance [5].

1.2 MOTIVATION

Slow adoption rates of the latest TLS versions is worrisome since it leaves users open to attacks. The authors of [2] show that around 45% of global websites are significantly vulnerable to exploits. This could easily be prevented by upgrading to the most available TLS version and disallowing future TLS downgrades. This report provides an overview of TLS 1.3 in hopes that developers will adopt the security standard as it becomes available and incentivize their platform becoming resistant to exploits, prevent future attacks, and reap the performance benefits of new security standards. Certainly the benefits of TLS 1.3 translate directly into mobile performance advantages.

1.3 STAKEHOLDERS

By observing the major stakeholders of mobile application security, one could get a better sense of the significance.

1.3.1 Government

Routinely the US government publishes material on mobile security [3]. They acknowledge the need for cryptography suitable for mobile devices. TLS 1.3 pushes the initiative further to meet the security requirements held by the government.

1.3.2 Mobile Application Developers

App developers need to keep their software secure and become familiar with security standards. Many of the mobile security exploits occur when software connects to network communications. On major mobile platforms, e.g., Apple and Android marketplaces, there are checkpoints in place to mitigate vulnerabilities, but some fall through the cracks. Banks and healthcare providers have additional requirements beyond typical internet security that must be followed. This goes to reinforce the software developers' responsibility to secure their software.

1.3.3 Hardware Vendors

Hardware vendors have it in their interest to make sure that the devices they sell are secure. Major technology vendors have an active role in setting the internet standards at the Internet Engineering Task Force (IETF). This is the same agency that adopts security standards such as TLS.

1.3.4 Cell Phone Carriers

Cell phone carriers like AT&T and Verizon consistently push device updates to protect against critical vulnerabilities. This is important because weak points can expose networks, and prevent availability of services.

1.3.5 End Users

Much of the burden for security lies with the end user. Many users have severely limited knowledge with security. They make bad decisions by going to inappropriate websites, downloading malicious applications, ignoring security warnings, and piggybacking on open wireless networks. The best practice for end

users is to ensure their devices are kept up-to-date with security from vendors and cell phone carriers.

1.4 STRUCTURE OF THIS REPORT

This report looks at internet application security for mobile devices. Section 2 is a technical overview of internet security and the underlying mechanics of TLS that specifically benefit mobile devices. In addition, the report looks at recent security exploits and how these exploits shape the current internet security protocols and mobile devices. Section 3 is a literature review of the current research into mobile application security. Section 4 is an analysis of the current security protocols in place for various mobile vendors and the performance implications on mobile devices.

2.0 TECHNICAL OVERVIEW

2.0 TECHNICAL OVERVIEW

The purpose of the technical overview section is to provide a broad picture of TLS and the exploits that are prominent for mobile devices. The current specification of the entire TLS 1.3 protocol working draft can be found at the following website: https://tools.ietf.org/html/draft-ietf-tls-tls13. Note that since TLS 1.3 is in a draft state, some features can change over time until the finalized version. Considering this, the report in itself is to showcase the possibilities of future security ideas and analyze the potential impacts on mobile computing.

Mobile devices described in the report are mobile cellular phones, smartphones, smartwatches, tablets, laptops, and other small form-factor computers. The mobile platform is generally optimized for a portable experience, and that includes compact and limited computer functionality, minimalistic UI, a plethora of wireless technologies, on demand services, and high availability. The main resource limitations of mobile devices are processor speed, local storage and memory size, network bandwidth and latency, and power consumption. All of these limitations have made it complex for obtainable security features. SSL/TLS is the backbone of the network security featured in computers and mobile devices.

2.1 SSL/TLS

Secure Socket Layer, SSL, is a toolset of cryptographic protocols that encrypts data through a communication network between a client and server. The main purpose of SSL is to provide confidential communication channels for users and ensure the integrity of information during data transmission. The tools negotiate the specific cryptographic protocol for public key exchange, authenticate the peers, specify the encryption method for exchange data and compression, and finally provide a hash method for the signature of the data. Note that both sides of the communication endpoints need SSL technology available to establish secure communications.

Netscape originally developed SSL which has now evolved into an internet standard called Transport Layer Security or TLS. Every day people use SSL when they are on the internet, using web apps and while utilizing their mobile devices. If a web browser internet page starts with HTTPS, then this internet connection is established by an SSL session. For example, https://www.google.com is a webpage that is using SSL shown in Figure 1.

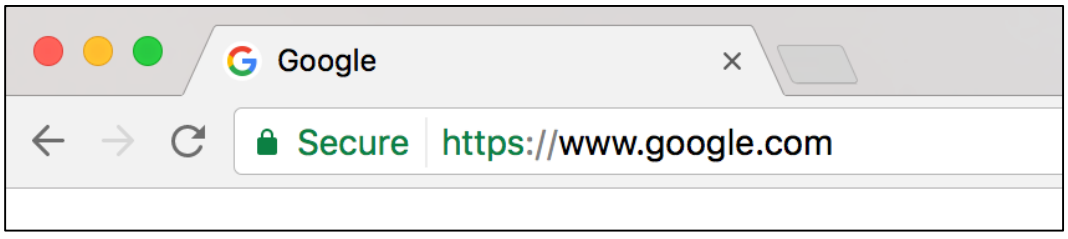

Figure 1: Example of HTTPS in Chrome Web Browser

TLS can be broken down into these basic steps:

1. The client initiates a ‘hello’ to a corresponding server, and the server responds with a ‘hello’ acknowledgement as well.
2. The client and server then negotiate the cryptographic protocols.
3. The client and server then generate a secret key.
4. To authenticate the server, and in some cases the client, the peers will verify a certificate that is trusted by a neutral party, a certificate authority (CA).
5. Once the server and client have been verified, secure data exchange is then possible.
6. Prior to messages transmission, a hash of the data with the sender’s public key will be attached to the message. The hash of the message is to verify the integrity of the message, known as signing or signature of the data. The final step is a symmetric encryption of the data with the secret key generated in step 3.
7. The message receiver then decrypts the data with the secret key in step 3 and can verify the hash of the message with the sender’s public key.
8. If something happens to the connection during the process, alerts will terminate the current secure communications.

TLS 1.2 is the most recent The Internet Engineering Task Force (IETF) approved version, which is described at length in IETF RFC 5246 [4]. The current rework TLS 1.3 has moderately different indicated goals than the previous version.

The IETF working charter describes the key goals of TLS 1.3 development as follows [5]:

- Encrypt as much of the handshake as possible
- Reduce handshake latency while still maintaining security features
- Address known security weaknesses
- Reevaluate handshake contents
- Improved Mobile CPU and network throughput

The goals can be broken down into two main areas: enhance security, and improve performance. Looking at the current state of TLS 1.3, a majority of the security and performance improvement is done in the TLS handshake protocol.

The TLS standard features two layered components, the record and the handshake protocol. These protocols are used to establish a transport protocol, TCP, communication channel, and allow for secure and reliable connection. The handshake protocol sets up the initial phase of communications between peers, and determines the key and type of cryptography for the session. On transition to the exchange data phase, the record protocol secures the session using the key determined in the handshake. The record protocol encapsulates all data encryption, breaks up data into manageable block sizes and then hashes the blocks of data. Then the data is passed along to the TCP connection. The record and handshake protocols have revisions in TLS 1.3. This report focuses on the specification of cryptography,

and the handshake protocol. Both specifications allow for improved latency over TLS 1.2 thus improving performance for mobile devices.

2.2 TLS HANDSHAKE

The handshake protocol establishes the first interaction between the client and server or application endpoints. A "hello" message back and forth between client and server starts the secure mechanism. Once secure communication is established, the protocol allows for data to be transmitted in the application layer. The handshake of TLS has the most latency because of the processing time needed to determine TLS version, key exchange method, cryptography, and authentication.

2.2.1 Hello Phase

The 'hello' phase is the initial encounter between client and server in the handshake protocol. The client sends a "Client Hello" message to the server, along with a random number and client time. The client's known cipher suite and key exchange is also passed along. Once the server receives the "Client Hello," the server responds by sending its own "Server Hello," random number, and server time, along with its supported cipher suite, key share, certificate with signature, and finally a Finished signal. The client can proceed to data exchange at this point. This is considered One Round-Trip Time or 1-RTT. In TLS 1.2 the "Client Hello" and cipher suite is first passed back and forth between the client and server, followed

by a key exchange, then another Finished signal. This is considered 2-RTT for TLS 1.2. The time saved is extremely valuable. Figures 2 and 3 demonstrates the procedure for 1-RTT and 2-RTT respectively.

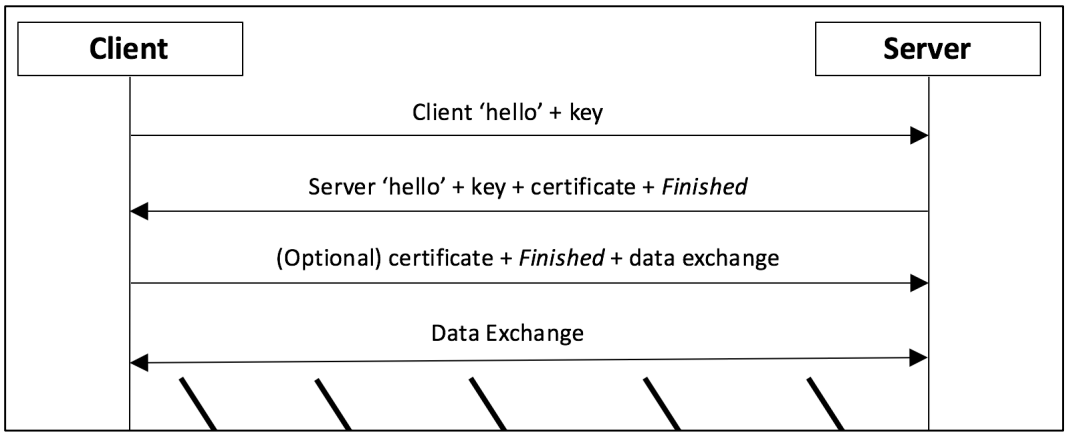

Figure 2: Full Handshake Protocol in TLS 1.3 with 1-RTT

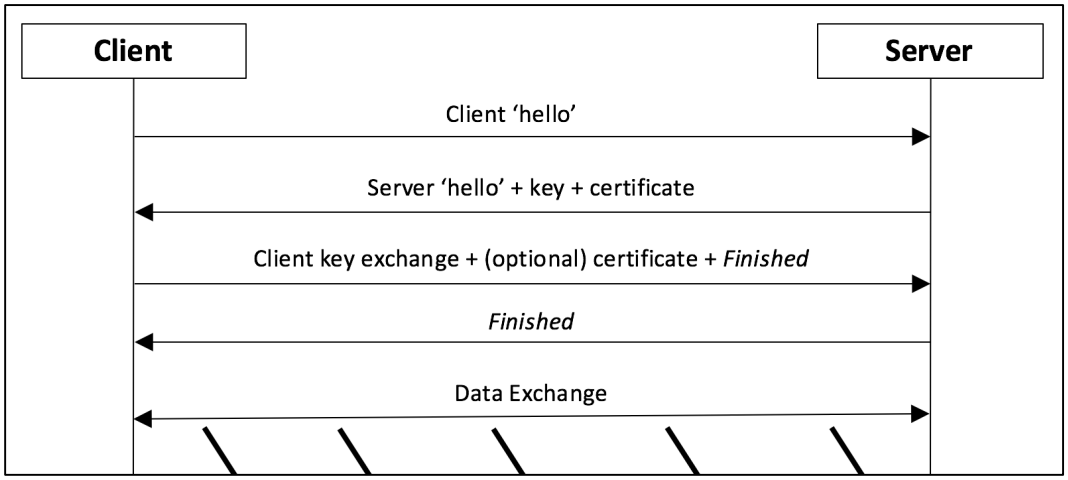

Figure 3: Full Handshake Protocol in TLS 1.2 with 2-RTT

The time saved on a connection that was established in the recent past is even better under TLS 1.3. There are several methods, but I only explain the basic

case. In TLS 1.3 there is 0-RTT for a resumption or session resumed connection. The client sends a "Client Hello" message with the previous shared key (PSK), then early data like an HTTP request is transmitted to the server; this process all transpires within the first flight message. Once the server receives the initial message, then it sends a "Server Hello," key exchange, Finished, then services the HTTP requests. 0-RTT is demonstrated in Figure 4.

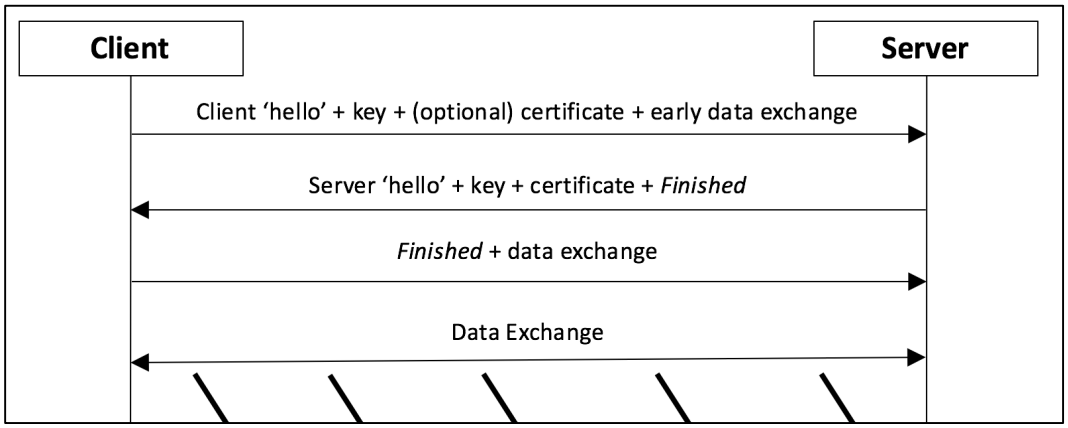

Figure 4: Resumption Handshake Protocol in TLS 1.3 with 0-RTT

2.2.2 Key Exchange

Diffie-Hellman (D-H) and Rivest-Shamir-Adleman (RSA) algorithms are the two main available methods of key exchange in the handshake phase for TLS connections. The current draft of TLS 1.3 requires Diffie-Hellman D-H key exchange, since RSA does not provide guarantees for forward secrecy. Perfect forward secrecy is a desired property of communications where past sessions are considered secure even if secure keys are leaked. The only method of breaking

forward secrecy is with brute force methods. One scenario is if a resumption connection uses a D-H key exchange, then it provides protection against forward secrecy. For early data transmission conditions, this can be dangerous, so HTTP requests are limited to idempotent REST lookups. Idempotent REST API's constraint certain types of HTTP requests that can be done prior to authentication.

D-H was the first public key cryptography algorithm that was developed by Ralph Merkle, Whitfield Diffie, and Martin Hellman, in 1976. Soon thereafter, in 1983, RSA was patented by Ron Rivest, Adi Shamir and Leonard Adleman. RSA has been the most dominant public key encryption mechanism, while D-H has been the most widely used key exchange method. However, in TLS 1.2 and prior versions, RSA key handshake has been the most popular and preferred method incorporated into the system [4].

Here is how the RSA key exchange works in TLS: The client generates a random number, which is encrypted with the public key of the server. The result is the secret key for the session phase. Only after the server's private key decrypts the secret key can data exchange happen. If authentication is required for the client, the server generates the secret key with the client's public key and a server random number, and the process of authentication repeats. For RSA to work properly, prior knowledge of the server or client public key is required for the key exchange.

One problem with RSA handshake key exchange is that it does not provide forward secrecy. If someone had access to previous stored session keys, then prior

data exchange is vulnerable to information loss. Keep in mind that a desirable property is confidentiality and is most crucial with archived data exchange in the future. D-H key exchange does not require prior knowledge of the two peers to start the handshake. With the invention of TLS 1.3, the IETF has chosen D-H to become the primary key exchange method, and has deprecated the RSA key exchange.

The D-H key exchange in TLS is as follows: the server generates a secret key by applying D-H algorithm, followed by a message with the server's public CA certificate. The client can then authenticate the server based on the mathematical properties of D-H and verify the certificate. If the client needs to be authenticated, then the server can request that the client apply the D-H algorithm with a random number to verify the client's CA certificate.

In TLS 1.3, Diffie-Hellman with random short-lived session tickets for a client and server key exchange handshake is highly recommended and currently the default setting. This is known as Ephemeral Diffie-Hellman (DHE). Note DHE always provides Perfect Forward Secrecy (PFS). Fixed Diffie-Hellman is an alternative D-H variant where the public key is included with the CA certificates. The same private key is used in Fixed D-H handshake, as opposed to new keys for DHE. It was widely available in TLS 1.2 and below, but has been replaced with DHE since it was susceptible to attacks with small key sizes. Anonymous D-H, the third D-H handshake, available in previous TLS versions is no longer

recommended since it does not authenticate peers nor protects against man-in-the-middle attack.

In May 2006, Elliptic Curve Cryptography was added to TLS 1.0 and 1.1, as specified in RFC 4492 [7]. The introduction sentence reads “Elliptic Curve Cryptography (ECC) is emerging as an attractive public-key crypto-system, in particular for mobile (i.e., wireless) environments” [7]. The developers of TLS explored faster cryptographic algorithms for the purpose of mobile devices prior to wide use. Recall that iPhones, which boosted the mobile platform market, were initially released in 2007. The IETF sensed that ECC would be best suited for constrained devices. Still despite TLS inclusion of ECC, RSA has been the most dominant protocol relied upon by TLS users. With TLS 1.3 the idea is to favor and recommend a handshake that uses ECC in the D-H handshake. The determination behind this is primarily that ECC provides equivalent security as does RSA but with smaller key sizes, as shown in Table 1 [6]. On mobile devices RSA impacts CPU utilization and memory storage. It is worth noting that ECC can be used as a full cryptography workload similar to RSA. ECC can be used for encryption, signing certificates, message authentication, and currently the most popular mechanism in the D-H key exchange.

Symmetric	ECC	DH/RSA/DSA
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Table 1: Cryptography Key Length in Bits [6]

Diffie-Hellman with ECC (ECDH) is a feature that can provide key exchange for the server and the client handshake phase. Instead of using discrete logarithm in D-H key exchange, ECC uses the mathematical operation of elliptic curves. The elliptic curve equation and graph of an example elliptic curve is as follows:

$y^2 = x^3 + ax + b$

Equation 1: Elliptic Curve Equation

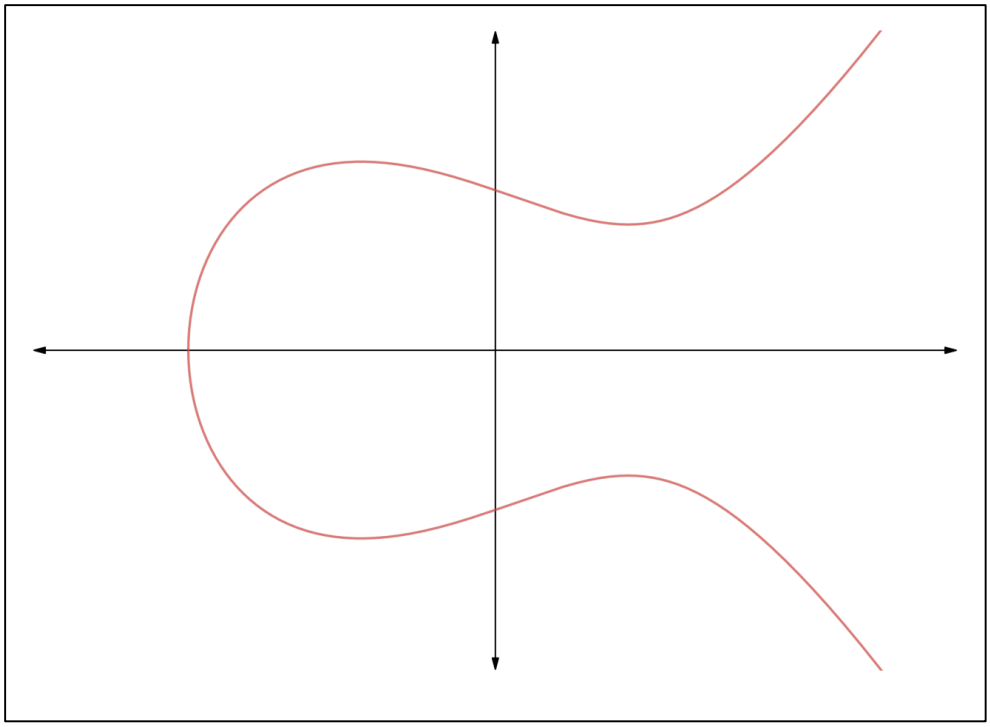

Figure 5: Example Plot of an Elliptic Curve

2.2.3 Elliptic Curve Diffie-Hellman

Here is how ECDH works in TLS: The client and server agree on a public elliptic curve $E(a,b)$ in the prime finite field Z_p. IETF has a list of popular curves that are recommended for TLS 1.3, namely [8]:

X25519: $y^2 = x^3 + 486662x^2 + x$ on finite field $2^{255} - 19$

and X448: $y^2 = x^3 + 156326\,x^2 + x$ on finite field $2^{448} - 2^{224} - 1$.

After the selection of the primes, both peers will select a base point on the curve, $B = (x_1, y_1)$ The client will then choose a random number, $n_C < |E|$ and

calculate public key $P_C = n_C \times B$. The server does the same calculation, choosing a random number, $n_S < |E|$ and calculate $P_S = n_S \times B$. The public keys generated are the "client key exchange" and "server key exchange" in the handshake. Next, the client and server send their public keys to each other and the secret key P_k, is then calculated using this formula: $P_k = n_C \times n_S \times B = P_S \times n_C = P_C \times n_S$. The values of the secret and public keys are points on the *E*. An eavesdropper would never be able to recreate the secret key because it is computationally infeasible to calculate P_k when given only P_C and P_S factors. The attacker would have to know the random values that were generated by the client or server. Since the key size is smaller compared to RSA, ECDH is an ideal key exchange mechanism for the mobile platform.

2.3 TLS ATTACKS

Attacks target vulnerabilities that were not identified prior to launching software to mass availability, especially on mobile devices. Even when flaws are identified is not always easy to close loopholes. However, at times vulnerability fixes are not complicated and could be done by disabling certain features and removing backward compatibility. It takes years of research and development to improve protocols and create better impenetrable security of information.

In this section, examples of known TLS attacks are listed. Many of the changes in TLS 1.3 have been largely influenced by the following man-in-the-middle (MitM) attacks. MitM attacks are so prevalent with mobile devices because end users accept the risks of joining public networks, to reduce bandwidth costs. The public networks allow for rerouting of services through private servers masquerading as friendly hotspots.

2.3.1 BEAST

BEAST (Browser Exploit Against SSL/TLS) is an attack that targets out-of-date SSL/TLS versions, namely TLS 1.0 and SSL 3.0. If an attacker is allowed access to HTTP browser cookies with a MitM attack, then an attacker can theoretically obtain authentication session tokens. These tokens can then be reused to gain access to the secure webpages. BEAST has only been displayed in demonstrations by researchers, and can be mitigated by upgrading TLS versions [9]. Note that mobile web browsers are all susceptible to desktop browser exploits.

2.3.2 CRIME

CRIME (Compression Ratio Info-leak Made Easy) is another MitM attack used to guess the session tokens when compression method is used in TLS [10]. An attacker can gain control of HTTPS requests and hijack message headers making brute force estimation on authentication tokens. Disabling compression altogether

in TLS is the only suitable solution. Currently compression is deprecated and not publicly recommended. Note that not using compression with TLS will slow down network traffic, this exploit has been known publicly for a few years. Therefore, this is not a mobile performance advantage. The only possible scenario is using compression with anonymous D-H with no authentication, presenting no need for security in that matter.

2.3.3 Extended CRIME: TIME

TIME (Time Infoleak Made Easy) MitM attack is similar model to the compression CRIME attack [11]. Instead of attacking HTTP requests, the attack focuses on HTTP response, which is a majority of HTTP interaction. As with CRIME, disabling compression is the only solution to a TIME attack.

2.3.4 BREACH

BREACH (Browser Reconnaissance & Exfiltration via Adaptive Compression of Hypertext) MitM attack is similar to the CRIME attack [12]. Rather than an attack on TLS compression, this attack exploits HTTP compression. The attack can force a client to disable SSL/TLS altogether and can view data in plaintext. There is no possible solution to this attack. So mobile devices have no option to use compression for an increased network throughput.

2.3.5 POODLE

POODLE (Padding Oracle On Downgraded Legacy Encryption) is a MitM attack that relies on clients downgrading SSL/TLS to obsolete SSL version 3.0 [13]. It exploits security flaws found in the older protocols. The best practice to avoid this attack is to prevent applications and web browsers from using SSL 3.0 and disabling downgrading. There are a lot of vendors now that require applications to support higher versions and specifically disable lower versions of TLS. However much of the internet is still vulnerable to this type of attack.

2.3.6 Logjam

Logjam is another MitM attack that relies on TLS downgrade. Researchers found that servers were using the most common 512-bit prime numbers to generate secret keys. [14] showed that 92% of HTTP servers with 512-bit D-H support used two distinct primes. This led to a simplified cryptanalysis search of the key information, and making the two primes unsafe. TLS now recommends large key size for D-H. Also this is a main reason reason that IETF supports a move to ECDH with TLS 1.3.

3.0 Literature Review

The technical overview purpose was to introduce the background of TLS 1.3 and improvements being made to overcome limitations in previous versions. The literature review gives an overview of the research that is influencing TLS for mobile devices.

Google's extensive knowledge of web traffic, led them to experiment with Quick UDP Internet Connections (QUIC) protocol. As the name implies, internet traffic routes through the datagram protocol in the transport layer, rather than TCP. The primary goal of QUIC is to reduce network latency at the expense of bandwidth. Google introduced a 0-RTT mechanism that is the foundation that TLS 1.3 0-RTT is built [15].

The OPTLS (OPTimized and/or for One-Point-Three TLS) framework provides design contributions to TLS 1.3. OPTLS provided a handshake rework from TLS 1.2, and added the observations from QUIC 0-RTT mechanism. The protocol introduced four separate handshakes that offer 1-RTT for new connections and 0-RTT for resumed connections. The protocol does not protect against MitM, or forward secrecy, but allows for new methodologies that TLS 1.3 will be built upon [16].

The authors of [17] provide a survey of mobile web browsers HTTPS security indications to users within the browser. They use the World Wide Web

Consortium (W3C) as a best practice for browser security interface, and compare those recommendations to that on mobile web browsers. They find that many mobile devices lack the presence of security indicators that is found in traditional desktop browsers. Most of the mobile browsers tested provide no warning to the user of invalid server certificates, indicating a potential compromised website.

Researchers present AndroSSL [18], a tool to test against TLS vulnerabilities on Android OS. The authors recognize the incompetence of developers understanding of security. AndroSSL demonstrates protections against many of the MitM attacks on TLS, and can warn developers to improve their applications. Today the main mobile applications markets provide the same level of protection offered with this tool.

To further emphasize the inability of developers' security knowledge, the authors of [19] do a full scale security study on free applications that are on the Google Play store. Some developers unknowingly leave holes in software, as in allowing exposed interprocess communication between application. Many developers ignore improper TLS warnings, and proceed to vulnerable HTTP websites. This can have severe consequences for the mobile experience.

4.0 Analysis

4.1 DIRECTION

With the mobile internet browsing outpacing desktops now [1], there is a major shift in the focus to address the limitations of mobile security. Proactive standardization is always the best practice for tackling future threats.

The protections currently in place to provide the best mobile security are sandboxing applications, security policies for applications sharing, application scanning on the OS stores, and secure communications. Sandboxing is the manner of isolating applications and restricting access to the applications own private memory space. This is the practice that is featured on all mobile devices today. Application sharing is where information can be exchanged between applications. Developers need to be aware of this, and open when only necessary. Manufacturers typically restrict expandable storage on mobile devices, so the vast majority of security issues come from outside of the device. Application developers need to be aware of weaknesses of code that attackers can exploit, and the developers must utilize TLS correctly when opening network ports on the device. Once an application is published to app stores, the stores have mechanisms in place to monitor applications for potential security concerns. This is not exactly a thorough process. Therefore, OS developers need to be proactive in security research to be one step ahead of hackers.

The current research areas on mobile security plays an important role for proactive threat protection. In general, these are some research areas that I have seen that are prevalent on mobile security:

- Various connectivity protection, e.g. Bluetooth, cellular, wireless networks
- Prevention methods to limit the application privileges and accesses of devices
- Mobile web browser UI limitations to highlight malicious internet content
- Creating dynamic security policies that better suits mobile computing
- Tools for developers to identify applications that are not compliant with security standards
- Improved throughput while maintaining rigorous security standards
- Agents to perform statistical analysis to pinpoint program abnormalities
- Offloading security duties to cloud based services

4.2 USABILITY: TLS MOBILE INTERNET SECURITY IN PRACTICE

In certain aspects, TLS in mobile devices is slightly different than that of typical desktop internet browsing, desktop software, and web based applications, even when using the web browser on an iPhone or Android. TLS is transparent on desktop internet web browsers, with a lock displayed on the webpage when enabled. The lock on the web address will specify the level of TLS and corresponding encryption methods. Some mobile browsers, such as Safari and

Chrome, the HTTPS lock is visible but provides no information about TLS security. Moreover, on mobile applications, TLS is embedded inside of the applications, and to normal users there is not a way to determine if secure communications are established other than assuming that the application is properly secure from the developer. A problematic situation is that mobile OS's cannot determine if any application needs secure communication, leaving the unsuspecting user vulnerable. Unless the application specifically denies connection when requested, it is likely the application proceeds with an unsecure connection. Apple has taken the guess work out of this dilemma with App Transport Security.

4.2.1 Apple iOS

App Transport Security (ATS) is Apple's security standard across the entire mobile device platform, including all iOS 9.0 (and above) and macOS 10.11. ATS is primarily a rebranded version of TLS 1.2 with a set of APIs available to developers. At the beginning of 2017, Apple mandated a strict requirement that all applications connecting to the internet use ATS, thus always providing end-to-end secure communications. ATS also resolves some of the troublesome unsecure cipher suites from TLS 1.2, namely RC4 and SHA-1, Both of which have been deprecated by IETF and no longer available in TLS 1.3 [20].

Apple is touting what they call "strong cryptography" in ATS [20], mainly referencing larger secret keys for the TLS session phase of data transfer, and the

prevention of downgrading to older TLS versions. AES encryption now requires at least 256-bit keys since AES with smaller keys are insecure and prone to attacks. SHA with 256-bit keys (SHA-2) is now available on Apple devices and enforced by default regarding data integrity. To protect users from future compromises of secret keys leaks, ATS recommends D-H key exchange with ECC. The move to ECDHE is the same that IETF has demonstrated for perfect forward secrecy and that Apple has publicly recommended for key exchange. ATS does allows for the use of RSA certificates along with Apple recommended ECDSA certificates to authentication peers.

Another significant security feature of ATS is the added Certificate Transparency. Certificate Transparency has three main services: logging of certificate activities, actively monitoring certificates, and auditing by certificate authorities (CA) [21]. Apple also monitors certificates in real-time to catch rogue activities. Therefore, application developers are forced to use a trusted CA. IETF has set up a working group for Certificate Transparency, and want this to become the standard practice for SSL/TLS certificates.

Currently there is no indication from Apple whether ATS will have TLS 1.3 features at the launch, but Apple has taken measures to easily transition into this version of TLS. Apple has updated the iOS security framework or Security Transport API, to invoke TLS sessions. Applications that connect to the internet now must use NSURLConnection and its successor NSURLSession.

NSURLSession is a class that allows for secure connection to URLs from iOS devices, basically HTTP and HTTPS requests.

The other API, WebView, displays HTTP webpages inside of iOS applications. WebView is an extremely powerful API, because many websites require paid API services that developers might not afford or do not allow API access. For example, public APIs allow for fetching and posting of data, and this can be directly used in an embedded application. When access to the public API is no longer available, then usually the only way to gather and display data is within a web browser. In this situation, WebView can take a snippet of a website and display that information in an application exactly how the webpage would load in a browser. This is the only situation whereby Apple allows ATS strict security to be turned off. Specifically, TLS does not need to be enabled between peers. When a new WebView application is uploaded to the Apple App Store, and ATS is turned off, this triggers features that will notify Apple. Then the developers need to justify why ATS is disabled, and request an exemption. These are the main features of Security Transport API and interaction with TLS.

A side-effect of ATS is that servers are now required to support TLS 1.2 and disable less than adequate security standards when connecting with ATS. Other platforms like desktop, IoT, and Android phones now benefit from the security feature.

Looking at the current industry, servers can be susceptible to downgrade attack due to inadequate upgraded applications and servers. Apple is forcing developers and technology maintainers to upgrade their applications and servers. This benefits not only iOS users, but desktop users, and other users of mobile platforms.

4.2.2 Google Android

Developers have the luxury of simple and rapid deployment of software using Android. The hefty entry price for iOS developers' license makes free Android development a bit more attractive as well, which translates into anyone having clearance to develop on the Android platform and publish content to numerous application stores. Google's approach to security is less restrictive than Apple which can leave end users vulnerable to shoddy development practices. It is not the responsibility of Google, or Apple for that matter, to determine if developers are using untrusted standards in applications that provide internet security. However, TLS is available in their development modules.

For Android applications that require HTTPS, Google offers the two APIs: HttpsURLConnection, SSLSocket. These are merely suggestions, which is a stark contrast with what Apple requires. Google Android offers WebView, similar to Apple, and they both have the same functionality and security concerns. While Google does publish an in-depth best practices manual online here:

https://developer.android.com/training/articles/security-ssl.html, WebView tends to leave the developer exposed if they have no knowledge of how to handle security holes. Google's security approach is that they provide a backend service to double check for vulnerabilities once a developer publishes content to the application store.

The SafetyNet Attestation API and SafetyNet Safe Browsing API provide automated security services if developers add this to their applications. With these APIs, developers have the option to test HTTP URLs against threats and analyze applications for quality security practices. Not only can the APIs test against the client applications, but the APIs can also perform network security logic for backend servers. The services will verify the TLS security versions and acceptable cipher suites.

All of the mentioned APIs currently support TLS 1.2 and lower, and there is not enough information to determine when TLS 1.3 will become available on the platform. Google contributes heavily to IETF, so one could speculate that they would support TLS 1.3 early on in the release cycle.

As mentioned, Google's tactic to application security is different than that of Apple. One could argue that the Android OS does not provide a strict requirement, as iOS, but they do offer all the necessary tools. There is not a mandatory prerequisite for security, and that is why many exploits are prevalent on the application stores. End users should have major concerns while relying on Android OS to deliver the best security experience.

4.3 PERFORMANCE

The available cipher suites and encryption methods found in the TLS 1.3 draft proposal are already available in TLS 1.2 and lower versions. The major differences, as specified in the technical overview, are mainly removing exposed protocols and improving latency with newer connection mechanisms. In this section, I test the performance of RSA and D-H. Also I measure the implementation of 0-RTT resumption.

4.3.1 Test Environment

OpenSSL is an open-source implementation of SSL and TLS. It features the ability to set up secure connections between clients and servers. OpenSSL is the underlining security library available in Apple and Android mobile devices. BoringSSL is an open-source Google fork of OpenSSL that is found in the Chrome browser and Android devices [22]. BoringSSL is the toolkit used in this report for testing the TLS functionality, since it is one of the few prototypes available for TLS 1.3. A smartphone would have been more desirable for the environment; however, the implementations of TLS 1.3 are not available, and are beyond the scope of this report. An Apple MacBook is the device under test; for benchmarking the cipher suites and connection.

Here is the description of the test environment:

- Hardware: 2015 MacBook Pro
- OS: macOS Sierra 10.12.4
- CPU: 2.5 GHz Intel i7
- Memory: 16 GB DDR3

4.3.2 Case Study

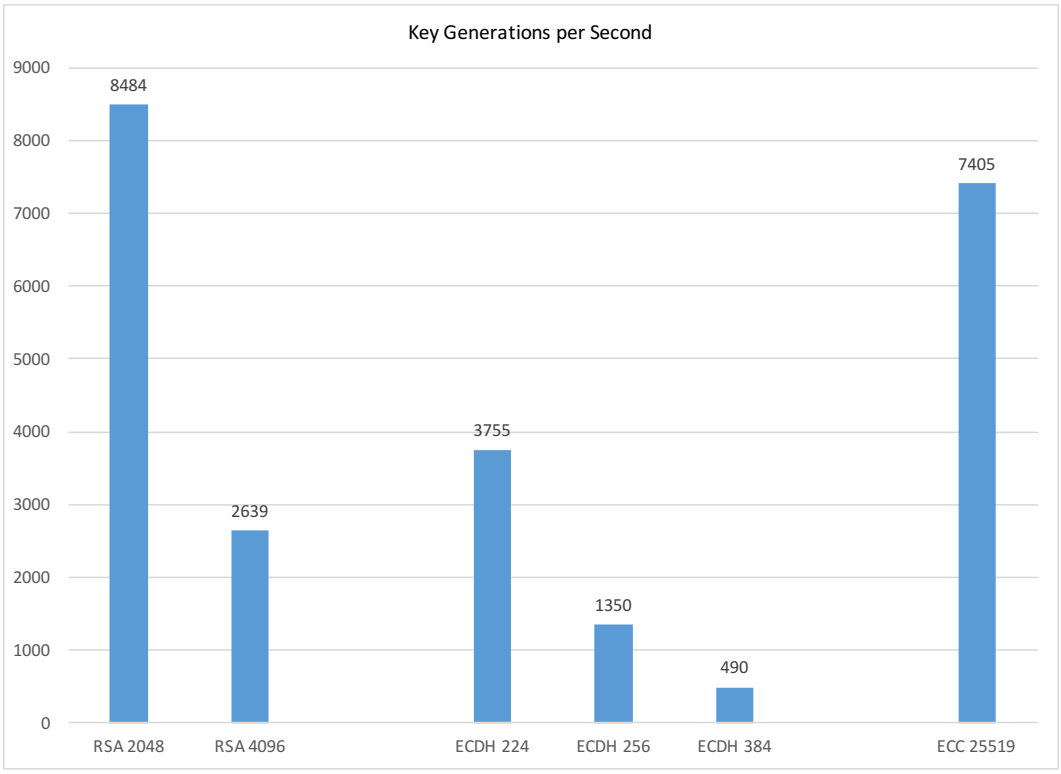

Figure 6: Number of Key Generations per Second in BoringSSL

BoringSSL's implementation of TLS 1.3 speed test was used to benchmark the key generation for RSA and ECDH in Figure 6. The key generation was taken

over a span of 100 second intervals. RSA with a key size 2048-bits outpaced an equivalent ECDH 224-bit key size, with a key generation of over 8000 operations a second. Performance severely degrades with an RSA 4096-bit key. For evaluation, ECDH key size 224-bit, 256-bit, and 384-bit are National Institute of Standards and Technology (NIST) standard ECC curves. The keys have the same comparable security strength as RSA 2048-bit, 3072-bit, 7680-bit keys respectively. Performance is considerably worse for ECDH.

RSA compared with particular ECC curves as 25519, show the power of ECC research. Clearly curve 25519 is better than the National Institute of Standards and Technology (NIST) ECC 256-bit specified key. Also for the same security as RSA 3072-bit key, it has quite similar performance at 7405 operations per second.

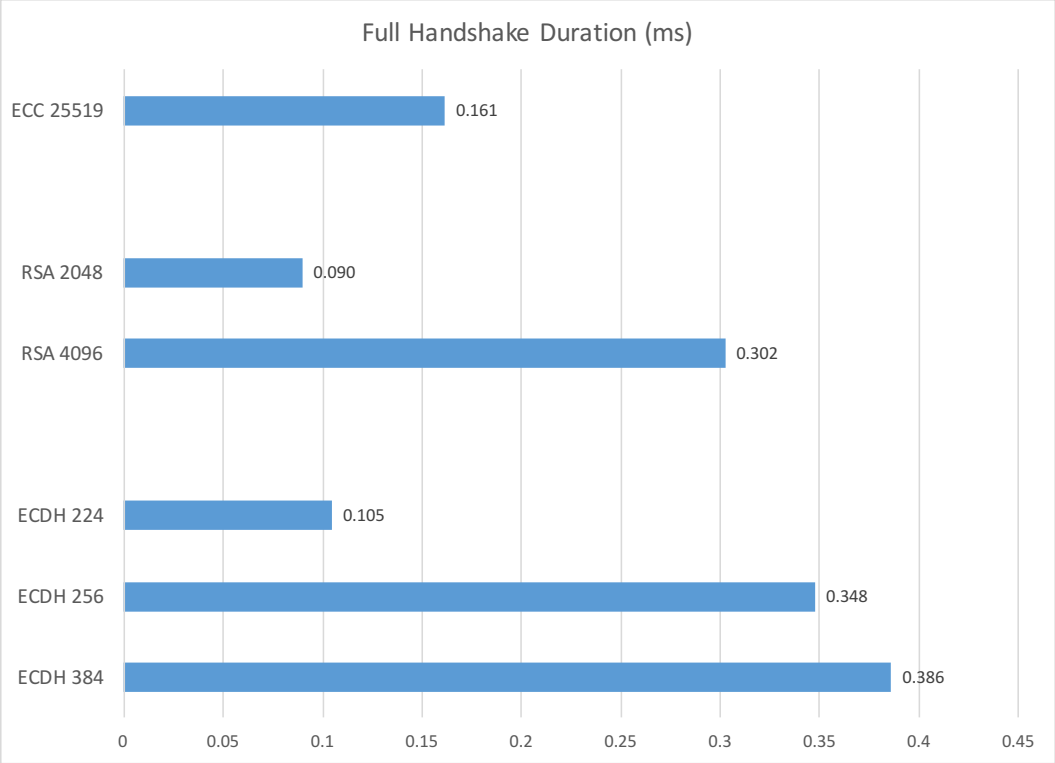

Figure 7: Full Handshake Duration (milliseconds) in BoringSSL

Next, BoringSSL's speed test was used to test the full handshake duration in milliseconds in Figure 7. Note that for authentication of ECC, DSA certificates were used, and RSA certificates for RSA authentication. Curve 25519 takes about half the time to complete the handshake for the same key size as NIST ECDH 256-key. Doubling the key size of RSA from 2048 to 4096-bit triples the handshake time. This is one of the reasons that TLS now recommends D-H. For NIST 256 and 384-bit key size, the performance was only difference by 0.038 milliseconds.

4.3.3 TLS Inoperability

Application: **BoringSSL**	**Security TLS 1.2** **Results (seconds)**	**Security level TLS 1.3** **Results (seconds)**
Server – client new connection	6.156s	5.727s
Server – client resumption	3.111s	2.394s
HTTPS curl	42.9s	43.9s

Table 2: Network Connection Comparison of TLS 1.2 and TLS 1.3

The purpose of TLS connection test was to demonstrate the current state of TLS 1.3 inoperability with current technology and comparison with TLS 1.2. Several different benchmarks were performed, and the result are displayed in Table 2.

First a server and client localhost session with BoringSSL was established with both TLS 1.2 and TLS 1.3. Four separate tests were performed. TLS 1.2 and TLS 1.3 with new connections, and using resumption session connection with both TLS versions. Each test result was summed for 1000 simulations. Resumption connection clearly performs better than a new connection. TLS 1.3 in both new and resumption connection was better than TLS 1.2.

Curl [23], the HTTP client command line tool, is another use case to test TLS 1.3. Curl was compiled with BoringSSL to reach HTTPS websites. 100 curl simulations to https://tls13.crypto.mozilla.org website server were tested. TLS 1.2 and TLS 1.3 have similar results around 43 seconds to perform 100 HTTPS requests.

4.3.3 Future Work

The case study is strictly limited due to the current state of TLS 1.3, which is not currently approved for internet security. Many vendors and the open source community have not invested in the latest security features thus far.

A more proper test when TLS 1.3 has general availability is to test in a lab setting with servers and mobile clients; with a dedicated wireless network bandwidth. A suitable test bench like Apache DayTrader would be sufficient [24]. Apache HTTP server can be the backend application server [25]. To simulate web traffic, Apache JMeter would be an appropriate tool [26].

5.0 Conclusion

There is no denying the dominance of mobile devices in today's world. Unfortunately, like most technologies, security is outdated and no longer meets the need of the mobile user. Since most users lack the knowledge to protect against attacks, governments and major vendors are on the hook to provide instruments that protect the consumer. Also since mobile devices require a multitude of communication networks to operate effectively, and so there are many opportunities of vulnerability. TLS 1.3 is a step in the right direction to provide necessary safeties.

The IETF started on TLS 1.3 in 2014, and the current draft is nearing release. With the rework of the Handshake protocol, mobile devices only gain to benefit given the limitations of the mobile platform. D-H and ECC have become popular because they provide better computational efficiencies and overall enhanced security features than the previous recommended RSA cryptography.

Apple App Transport Security requires all iOS and macOS applications that require internet to be at a minimum TLS 1.2 version, which proactively forces developers and IT maintainers to keep updated with TLS standard. This standard also necessitates that backend servers that service data and information to also comply with Apple's security requirement. Other vendors like Google provide developers with safeguards that allow for checking applications and backend servers against known weaknesses.

The performance benchmarks of this report provide a model or proof-of-concept of the capabilities of TLS 1.3 features. Remember that TLS 1.3 is in a draft

state, so future work is necessary to fully determine true security and performance impacts. Nevertheless, the results of this report demonstrate that TLS 1.3 positively benefit the future direction security, which only transforms into improved performance across the mobile platform.

Bibliography

[1] “Mobile and tablet internet usage exceeds desktop for first time worldwide.” StatCounter Global Stats, Nov. 2016. http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide. Accessed March 1, 2017.

[2] Oh, Sanghak, Eunsoo Kim, and Hyoungshick Kim. "Empirical analysis of SSL/TLS weaknesses in real websites: Who cares?."

[3] Quirolgico, Steve, et al. "Vetting the security of mobile applications." *NIST special publication* 800 (2015): 163.

[4] Dierks, T. et al. “The Transport Layer Security (TLS) Protocol Version 1.2.” The Internet Engineering Task Force (IETF), Aug. 2008, https://www.ietf.org/rfc/rfc5246.txt. Accessed March 1, 2017.

[5] Transport Layer Security (TLS): Charter for Working Group. The Internet Engineering Task Force (IETF), Jul. 2016, https://datatracker.ietf.org/wg/tls/charter. Accessed March 1, 2017.

[6] Gupta, Vipul, et al. "Performance analysis of elliptic curve cryptography for SSL." *Proceedings of the 1st ACM workshop on Wireless security*. ACM, 2002.

[7] Blake-Wilson, S. et al. “Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS).” The Internet Engineering Task Force (IETF): Network Working Group, May 2006, https://www.ietf.org/rfc/rfc4492.txt. Accessed March 1, 2017.

[8] Langley, A. et al. “Elliptic Curves for Security.” Internet Research Task Force (IRTF), Jan. 2016, https://tools.ietf.org/html/rfc7748. Accessed March 1, 2017.

[9] Duong, Thai, and Juliano Rizzo. "Here come the ⊕ ninjas." *Unpublished manuscript* 320 (2011).

[10] Duong, Thai, and Juliano Rizzo. "The CRIME attack." *Presentation at ekoparty Security Conference*. 2012.

[11] Be'ery, Tal, and Amichai Shulman. "A perfect crime? only time will tell." *Black Hat Europe* 2013 (2013).

[12] Prado, Angelo, Neal Harris, and Yoel Gluck. "Ssl, gone in 30 seconds-a breach beyond crime." *Black Hat USA* (2013).

[13] Möller, Bodo, Thai Duong, and Krzysztof Kotowicz. "This POODLE bites: exploiting the SSL 3.0 fallback." *Security Advisory* (2014).

[14] Adrian, David, et al. "Imperfect forward secrecy: How Diffie-Hellman fails in practice." *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[15] Roskind, Jim. "Quic: Design document and specification rational." (2015).

[16] Krawczyk, Hugo, and Hoeteck Wee. "The OPTLS protocol and TLS 1.3." *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016.

[17] Amrutkar, Chaitrali, Patrick Traynor, and Paul C. Van Oorschot. "An empirical evaluation of security indicators in mobile Web browsers." *IEEE Transactions on Mobile Computing* 14.5 (2015): 889-903.

[18] Gagnon, François, et al. "AndroSSL: A Platform to Test Android Applications Connection Security." *International Symposium on Foundations and Practice of Security*. Springer International Publishing, 2015.

[19] Mutchler, Patrick, et al. "A large-scale study of mobile web app security." *Mobile Security Techologies* (2015).

[20] Ballard, Lucia, and Cooper, Simon. "What's New in Security: Session 706." *Apple Worldwide Developer Conference 2016*.

[21] "How Certificate Transparency Works - Certificate Transparency." Google, https://www.certificate-transparency.org/how-ct-works. Accessed March 1, 2017.

[22] "BoringSSL." https://boringssl.googlesource.com. Accessed March 1, 2017.

[23] "Curl." https://curl.haxx.se. Accessed March 1, 2017.

[24] “Apache DayTrader” http://geronimo.apache.org/GMOxDOC20/daytrader.html. Accessed March 1, 2017.

[25] “Apache HTTP Server Project.” https://httpd.apache.org. Accessed March 1, 2017.

[26] “Apache JMeter.” http://jmeter.apache.org. Accessed March 1, 2017.

Vita

Vincent Davis was born in San Antonio, Texas. After fulfilling his military obligation, he attended community college at San Antonio College. Eventually he transferred into the University of Texas at Austin, where he received a Bachelor of Science in Electrical Engineering in May 2011. He works as a software engineer in Los Angeles, California. In August, 2014, he began graduate studies at the University of Texas at Austin.

Address: davis.vincent@utexas.edu

This report was typed by the author.